LP1:

Factorial.java

```
import java.math.BigInteger;
public interface Factorial extends java.rmi.Remote{
 public BigInteger factorial(int n) throws java.rmi.RemoteException;
}
```

FactorialClient.java

```
import java.rmi.Naming;
import java.net.MalformedURLException;
import java.rmi.NotBoundException;
import java.rmi.RemoteException;
public class FactorialClient {
 public static void main(String[] args) {
 try {
 Factorial fact = (Factorial) Naming.lookup("rmi://localhost/FactorialService");
 System.out.println(fact.factorial(30));
 } catch (MalformedURLException e) {
 System.out.println("Malformed URL: " + e);
 } catch (RemoteException e) {
 System.out.println("Remote Exception: " + e);
 } catch (NotBoundException e) {
 System.out.println("Not Bound Exception: " + e);
 }catch (ArithmeticException ae){
 System.out.println("Arithmetic Exception: "+ae);
 }
 } }
```

FactorialImpl.java

```
import java.math.BigInteger;
import java.rmi.RemoteException;

public class FactorialImpl extends java.rmi.server.UnicastRemoteObject implements
Factorial{
 protected FactorialImpl() throws RemoteException {
 super();
 }
  @Override

  public BigInteger factorial(int n) throws RemoteException {
    BigInteger fact = BigInteger.ONE;
 for (int i = 1; i <= n; i++) {
```

```
 fact = fact.multiply(BigInteger.valueOf(i));
 }
 return fact;
    }
}
```

FactorialServer.java

```
import java.rmi.Naming;
public class FactorialServer {
 public FactorialServer(){
 try {
 Factorial factorial = new FactorialImpl();
 Naming.rebind("rmi://localhost/FactorialService", factorial);
 }catch (Exception e){
 System.out.println("Exception: "+e);
 }
 }
 public static void main(String[] args) {
 new FactorialServer();
 } }
```

OUTPUT:

```
PS C:\Users\ IdeaProjects\DS_Assignment1> cd src
PS C:\Users\ IdeaProjects\DS_Assignment1\src> javac Factorial.java FactorialImpl.java
FactorialClient.java
FactorialServer.java
PS C:\Users\IdeaProjects\DS_Assignment1\src> rmic FactorialImpl
Warning: generation and use of skeletons and static stubs for JRMP
is deprecated. Skeletons are unnecessary, and static stubs have
been superseded by dynamically generated stubs. Users are
encouraged to migrate away from using rmic to generate skeletons and static
stubs. See the documentation for java.rmi.server.UnicastRemoteObject.
```

LP3:

Pr3.py

```
import numpy as np
import multiprocessing as mp

def calculate_sum(array):
    partial_sum = np.sum(array)
    return partial_sum
```

```
if __name__ == '__main__':
    num_processes = mp.cpu_count()  # Number of available processes/threads
    print("Enter size of array")
    array_size = int(input())  # Size of the input array
    array = np.arange(1, array_size + 1)  # Input array
    print("Input Array:", array)
    # Divide the array into equal chunks for each process
    chunk_size = array_size // num_processes
    chunks = [array[i:i+chunk_size] for i in range(0, array_size, chunk_size)]

    # Calculate partial sums using OpenMP
    with mp.Pool(processes=num_processes) as pool:
        partial_sums = pool.map(calculate_sum, chunks)

    # Display intermediate sums calculated at different processes
    for i, partial_sum in enumerate(partial_sums):
        print(f"Process {i}: Partial Sum = {partial_sum}")

    # Calculate the final sum by combining the partial sums
    total_sum = np.sum(partial_sums)

    print("Total Sum:", total_sum)
```

OUTPUT:

```
Enter size of array
150
Input Array: [ 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36
 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54
 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72
 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90
 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108
109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126
127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144
145 146 147 148 149 150]

Process 0: Partial Sum = 45
Process 1: Partial Sum = 126
Process 2: Partial Sum = 207
Process 3: Partial Sum = 288
Process 4: Partial Sum = 369
Process 5: Partial Sum = 450
Process 6: Partial Sum = 531
Process 7: Partial Sum = 612
Process 8: Partial Sum = 693
Process 9: Partial Sum = 774
Process 10: Partial Sum = 855
Process 11: Partial Sum = 936
Process 12: Partial Sum = 1017
Process 13: Partial Sum = 1098
Process 14: Partial Sum = 1179
Process 15: Partial Sum = 1260
Process 16: Partial Sum = 885
Total Sum: 11325
```

Total sum is the sum of process 0 to 16 i.e.
Process0+Process1+Process2+..+process16=11325 and not the sum of first 150 numbers

LP4:

Server.py

```
# Python3 program imitating a clock server

from functools import reduce
from dateutil import parser
import threading
import datetime
import socket
import time


# datastructure used to store client address and clock data
client_data = {}


''' nested thread function used to receive
     clock time from a connected client '''
def startReceivingClockTime(connector, address):

     while True:
          # receive clock time
          clock_time_string = connector.recv(1024).decode()
          clock_time = parser.parse(clock_time_string)
          clock_time_diff = datetime.datetime.now() - \
                                                   clock_time

          client_data[address] = {
                         "clock_time"    : clock_time,
                         "time_difference" : clock_time_diff,
                         "connector"     : connector
                         }

          print("Client Data updated with: "+ str(address),
                                                       end = "\n\n")
          time.sleep(5)


''' master thread function used to open portal for
     accepting clients over given port '''
def startConnecting(master_server):

     # fetch clock time at slaves / clients
     while True:
          # accepting a client / slave clock client
          master_slave_connector, addr = master_server.accept()
          slave_address = str(addr[0]) + ":" + str(addr[1])

          print(slave_address + " got connected successfully")

          current_thread = threading.Thread(
                              target = startReceivingClockTime,
                              args = (master_slave_connector,
                                                  slave_address, ))
          current_thread.start()


# subroutine function used to fetch average clock difference
def getAverageClockDiff():

     current_client_data = client_data.copy()

     time_difference_list = list(client['time_difference']
                                        for client_addr, client
                                             in client_data.items())


     sum_of_clock_difference = sum(time_difference_list, \
                                        datetime.timedelta(0, 0))

     average_clock_difference = sum_of_clock_difference \
                                                  / len(client_data)

     return average_clock_difference


''' master sync thread function used to generate
     cycles of clock synchronization in the network '''
def synchronizeAllClocks():

     while True:

          print("New synchronization cycle started.")
```

```
          print("Number of clients to be synchronized: " + \
                                   str(len(client_data)))

          if len(client_data) > 0:

               average_clock_difference = getAverageClockDiff()

               for client_addr, client in client_data.items():
                    try:
                         synchronized_time = \
                              datetime.datetime.now() + \
                                             average_clock_difference

                         client['connector'].send(str(
                                   synchronized_time).encode())

                    except Exception as e:
                         print("Something went wrong while " + \
                              "sending synchronized time " + \
                              "through " + str(client_addr))

          else :
               print("No client data." + \
                              " Synchronization not applicable.")

          print("\n\n")

          time.sleep(5)


# function used to initiate the Clock Server / Master Node
def initiateClockServer(port = 8080):

     master_server = socket.socket()
     master_server.setsockopt(socket.SOL_SOCKET,
                                   socket.SO_REUSEADDR, 1)

     print("Socket at master node created successfully\n")

     master_server.bind(('', port))

     # Start listening to requests
     master_server.listen(10)
     print("Clock server started...\n")

     # start making connections
     print("Starting to make connections...\n")
     master_thread = threading.Thread(
                         target = startConnecting,
                         args = (master_server, ))
     master_thread.start()

     # start synchronization
     print("Starting synchronization parallelly...\n")
     sync_thread = threading.Thread(
                         target = synchronizeAllClocks,
                         args = ())
     sync_thread.start()



# Driver function
if __name__ == '__main__':

     # Trigger the Clock Server
     initiateClockServer(port = 8080)
```

client.py

```
# Python3 program imitating a client process

from timeit import default_timer as timer
from dateutil import parser
import threading
import datetime
import socket
import time


# client thread function used to send time at client side
def startSendingTime(slave_client):

     while True:
          # provide server with clock time at the client
          slave_client.send(str(
                         datetime.datetime.now()).encode())

          print("Recent time sent successfully",
                                             end = "\n\n")
          time.sleep(5)


# client thread function used to receive synchronized time
def startReceivingTime(slave_client):

     while True:
          # receive data from the server
          Synchronized_time = parser.parse(
                              slave_client.recv(1024).decode())

          print("Synchronized time at the client is: " + \
                                             str(Synchronized_time),
                                             end = "\n\n")


# function used to Synchronize client process time
def initiateSlaveClient(port = 8080):

     slave_client = socket.socket()

     # connect to the clock server on local computer
     slave_client.connect(('127.0.0.1', port))

     # start sending time to server
     print("Starting to receive time from server\n")
     send_time_thread = threading.Thread(
                         target = startSendingTime,
                         args = (slave_client, ))
     send_time_thread.start()


     # start receiving synchronized from server
     print("Starting to receiving " + \
                              "synchronized time from server\n")
     receive_time_thread = threading.Thread(
                         target = startReceivingTime,
                         args = (slave_client, ))
     receive_time_thread.start()


# Driver function
if __name__ == '__main__':

     # initialize the Slave / Client
     initiateSlaveClient(port = 8080)
```

OUTPUT:

sever.py

```
New synchronization cycle started.
Number of clients to be synchronized: 0
No client data. Synchronization not applicable.
127.0.0.1:56656 got connected successfully
```

```
Client Data updated with: 127.0.0.1:56656
New synchronization cycle started.
Number of clients to be synchronized: 1
127.0.0.1:56664 got connected successfully
Client Data updated with: 127.0.0.1:56664
New synchronization cycle started.
Number of clients to be synchronized: 2
Client Data updated with: 127.0.0.1:56656
Client Data updated with: 127.0.0.1:56664
New synchronization cycle started.
Number of clients to be synchronized: 2
```

client1.py

```
Starting to receive time from serve
Starting to receiving synchronized time from server
Recent time sent successfully
Synchronized time at the client is: 2023-05-18 15:15:14.293920
```

client2.py

```
Starting to receive time from server
Starting to receiving synchronized time from server
Recent time sent successfully
Synchronized time at the client is: 2023-05-18 15:15:34.296732
```

LP5:

Client.java

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.Socket;

public class Client {
 private String serverAddress;
 private int serverPort;
 private BufferedReader reader;
 private PrintWriter writer;
 public Client(String serverAddress, int serverPort) {
```

```
 this.serverAddress = serverAddress;
 this.serverPort = serverPort;
 }
 public void connect() {
 try {
 Socket socket = new Socket(serverAddress, serverPort);
 System.out.println("Connected to server: " +
socket.getInetAddress().getHostAddress());
 reader = new BufferedReader(new InputStreamReader(socket.getInputStream()));
 writer = new PrintWriter(socket.getOutputStream(), true);
 // Start sending and receiving messages
 startMessageExchange();
 } catch (IOException e) {
 e.printStackTrace();
 }
 }

 public void startMessageExchange() {
 new Thread(() -> {
 try {
 while (true) {
 // Read message from server
 String message = reader.readLine();
 if (message != null) {
 System.out.println("Message received: " + message);
 }
 }
 } catch (IOException e) {
   e.printStackTrace();
   }
 }).start();

  try {
 BufferedReader consoleReader = new BufferedReader(new InputStreamReader(System.in));
 while (true) {
 // Read input from console
 String input = consoleReader.readLine();
 if (input != null) {
 // Send message to server
 writer.println(input);
 }
 }
 } catch (IOException e) {
 e.printStackTrace();
 }
 }
  public static void main(String[] args) {
 Client client = new Client("localhost", 8080);
 client.connect();
 } }
```

Mutualserver.java

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.ServerSocket;
import java.net.Socket;

public class MutualServer {
 private ServerSocket serverSocket;
 private Socket client1Socket;
 private Socket client2Socket;
 private BufferedReader client1Reader;
 private PrintWriter client1Writer;
 private BufferedReader client2Reader;
 private PrintWriter client2Writer;
 public MutualServer(int port) {
 try {
 serverSocket = new ServerSocket(port);
 System.out.println("Server started and listening on port " + port);
 } catch (IOException e) {
 e.printStackTrace();
 }
 }

  public void acceptConnections() {
 try {
   // Accept client 1 connection
 client1Socket = serverSocket.accept();
 System.out.println("Client 1 connected: " +
client1Socket.getInetAddress().getHostAddress());
 client1Reader = new BufferedReader(new
InputStreamReader(client1Socket.getInputStream()));
 client1Writer = new PrintWriter(client1Socket.getOutputStream(), true);

   // Accept client 2 connection
 client2Socket = serverSocket.accept();
 System.out.println("Client 2 connected: " +
client2Socket.getInetAddress().getHostAddress());
 client2Reader = new BufferedReader(new
InputStreamReader(client2Socket.getInputStream()));
 client2Writer = new PrintWriter(client2Socket.getOutputStream(), true);

   // Start message exchange
 startMessageExchange();
 } catch (IOException e) {
```

```
e.printStackTrace();
}
}

 public void startMessageExchange() {
new Thread(() -> {

  try {
while (true) {
// Read message from client 1
String messageFromClient1 = client1Reader.readLine();
if (messageFromClient1 != null) {
System.out.println("Message received from Client 1: " + messageFromClient1);
// Forward message to client 2
client2Writer.println(messageFromClient1);
}
}
} catch (IOException e) {
e.printStackTrace();
}
}).start()

try {
while (true) {
// Read message from client 2
String messageFromClient2 = client2Reader.readLine();
if (messageFromClient2 != null) {
System.out.println("Message received from Client 2: " + messageFromClient2);
// Forward message to client 1
client1Writer.println(messageFromClient2);
}
}
} catch (IOException e) {
e.printStackTrace();
}
}
public static void main(String[] args) {
MutualServer server = new MutualServer(8080);
server.acceptConnections();
} }
```

OUTPUT:

Create 2 client.java files viz. Client and Clients
Client will behave as client1 and Clients will behave as Client 2
For Clients public class Clients

First run Mutualserver.java then Client.java and last run Clients.java
All 3 in same folder

MutualServer.java

```
Server started and listening on port 8080
Client 1 connected: 127.0.0.1
Client 2 connected: 127.0.0.1
Message received from Client 1: Hello, Client 2
Message received from Client 2: Hello, Client 1
Message received from Client 1: This is Token Ring Algorithm for client2
Message received from Client 2: Thankyou client1
```

Client1

```
Connected to server: 127.0.0.1
 Hello, Client 2
Message received: Hello, Client 1
This is Token Ring Algorithm for client2
Message received: Thankyou client1
```

Client2

```
Connected to server: 127.0.0.1
Message received: Hello, Client 2
Hello, Client 1
Message received: This is Token Ring Algorithm for client2
Thankyou client1
```

LP6:

BullyAlgorithm.java

```java
import java.util.ArrayList;
import java.util.List;

public class BullyAlgorithm {
    public static void main(String[] args) {
        // Create nodes
        List<Node> nodes = new ArrayList<>();
        for (int i = 1; i <= 5; i++) {
            nodes.add(new Node(i));
        }
        // Node 3 initiates the election
        nodes.get(2).sendElectionMessage(nodes);
        // Node 1 crashes
        nodes.get(0).crash();
        // Node 4 initiates the election
        nodes.get(3).sendElectionMessage(nodes);
    } }
```

Node.java

```
import java.util.List;
class Node {
    private int nodeId;
    private boolean isActive = true;
    Node(int nodeId) {
        this.nodeId = nodeId;
    }
    void sendElectionMessage(List<Node> nodes) {
        System.out.println("Node " + nodeId + " sends Election message.");
        for (Node node : nodes) {
            if (node.nodeId > this.nodeId) {
                node.receiveElectionMessage(this);
            }
        }

        // If no higher node responds, this node becomes the leader

        System.out.println("Node " + nodeId + " becomes the leader.");
    }

    void receiveElectionMessage(Node sender) {
        System.out.println("Node " + nodeId + " received Election message from Node "
+ sender.nodeId + ".");
        if (isActive) {
            // Node is active, respond with an OK message
            System.out.println("Node " + nodeId + " sends OK message to Node " +
sender.nodeId + ".");
            sender.receiveOKMessage(this);
        }
    }

    void receiveOKMessage(Node sender) {
        System.out.println("Node " + nodeId + " received OK message from Node " +
sender.nodeId + ".");
        // Ignore the OK message since this node is not initiating the election
    }

    void crash() {
        System.out.println("Node " + nodeId + " crashes.");
        isActive = false;
    } }
```

OUTPUT:

Run BullyAlgorithm

```
Node 3 sends Election message.
Node 4 received Election message from Node 3.
Node 4 sends OK message to Node 3.
Node 3 received OK message from Node 4.
Node 5 received Election message from Node 3.
Node 5 sends OK message to Node 3.
Node 3 received OK message from Node 5.
Node 3 becomes the leader
Node 1 crashes.
Node 4 sends Election message.
Node 5 received Election message from Node 4.
Node 5 sends OK message to Node 4.
Node 4 received OK message from Node 5.
Node 4 becomes the leader.
Process finished with exit code 0
```

LP7:

Distributed_application.py

```
import requests
import multiprocessing
import os

def consume_web_service(app_id):
  url = f'http://127.0.0.1:5000/api/hello?app_id={app_id}'
  response = requests.get(url)
if response.status_code == 200:
  data = response.json()
  message = data['message']
  print(f'Response from App ID {app_id}:', message)

  else:
    print(f'Failed to fetch data from the web service for App ID {app_id}')
  if __name__ == '__main__':
    num_instances = 5 # Number of distributed application instances to run
  processes = []
for i in range(num_instances):
  process = multiprocessing.Process(target=consume_web_service, args=(i,))
  process.start()
  processes.append(process)

for process in processes:
  process.join()
```

Web_service.py

```
from flask import Flask, jsonify
app = Flask(__name__)
@app.route('/api/hello', methods=['GET'])
def hello():
  return jsonify(message='Hello, World!')

if __name__ == '__main__':
  app.run(debug=True)
```

OUTPUT:

web-service.py

```
* Serving Flask app 'web_service2'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a
production WSGI server
instead.
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
* Restarting with watchdog (windowsapi)
* Debugger is active!
* Debugger PIN: 375-964-059

127.0.0.1 - - [18/May/2023 15:28:57] "GET /api/hello?app_id=0 HTTP/1.1" 200 -
127.0.0.1 - - [18/May/2023 15:28:57] "GET /api/hello?app_id=1 HTTP/1.1" 200 -
127.0.0.1 - - [18/May/2023 15:28:57] "GET /api/hello?app_id=2 HTTP/1.1" 200 -
127.0.0.1 - - [18/May/2023 15:28:57] "GET /api/hello?app_id=3 HTTP/1.1" 200 -
127.0.0.1 - - [18/May/2023 15:28:57] "GET /api/hello?app_id=4 HTTP/1.1" 200 -
127.0.0.1 - - [18/May/2023 15:29:09] "GET /api/hello?app_id=1 HTTP/1.1" 200 -
127.0.0.1 - - [18/May/2023 15:29:09] "GET /api/hello?app_id=0 HTTP/1.1" 200 -
127.0.0.1 - - [18/May/2023 15:29:09] "GET /api/hello?app_id=2 HTTP/1.1" 200 -
127.0.0.1 - - [18/May/2023 15:29:09] "GET /api/hello?app_id=3 HTTP/1.1" 200 -
```

```
127.0.0.1 - - [18/May/2023 15:29:09] "GET /api/hello?app_id=4 HTTP/1.1" 200 -
```

distributed_application.py (App 1)

```
Response from App ID 0: Hello, World!
Response from App ID 1: Hello, World!
Response from App ID 2: Hello, World!
Response from App ID 3: Hello, World!
Response from App ID 4: Hello, World!
```

distributed_application.py (App 2)

```
Response from App ID 0: Hello, World!
Response from App ID 1: Hello, World!
Response from App ID 2: Hello, World!
Response from App ID 3: Hello, World!
Response from App ID 4: Hello, World!
```